ORACLE
Fusion Middleware
Forms

Oracle Forms
Standalone Launcher (FSAL)

Using and Securing Applications Running with FSAL - Rev.2

ORACLE WHITE PAPER | SEPTEMBER 2019

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

This document will further outline the intended use for Oracle Forms Standalone Launcher (FSAL). Security information and suggestions provided in this document are for example purposes only. It is your responsibility to understand the concepts outlined and properly test any implementation derived from this content. Misuse of the Oracle products or dependent technologies may result in the degradation of your system or application security. Be sure to refer to the included references to ensure you are properly using the product and any dependent technologies.

Table of Contents

Introduction

As browser vendors move away from supporting plug-ins, technologies like Oracle Forms will need alternative ways in which to run the applications developed with them. Beginning in version 12.2.1.0.0, Oracle Forms introduced one such alternative.

The Forms Standalone Launcher (FSAL) offers an alternative way for end-users to run Forms 12c applications. FSAL provides a browser-less, client/server-like interface. As a result of not using a browser, FSAL does not rely on the Java Deployment technologies (e.g. Plug-in, Web Start, etc) and therefore is not dependent on a browser. However, it does require Java on the end-user machine.

This paper will offer basic usage information, as well as some tips on ensuring that your applications run as securely as possible. Refer to the official documentation for more information.

This paper will assume the use of Oracle Forms 12.2.1.4.0. Some features described in this document may not be available in earlier releases of version 12.2.1.x; however most concepts will still apply to any 12.2.1 release. In the context of this paper, all Java references are assumed to be relative to Java version 8u211+, unless noted otherwise.

Understanding the Forms Standalone Launcher

The Forms Standalone Launcher (FSAL) mimics several client/server concepts, however with some modern flare. In the early days of Oracle Forms client/server, end-user machines required the installation of the Oracle Forms runtime software, as well as the application(s) planned to be run on that machine. This was often a cumbersome process and consumed significant disk space. Having the software and application installed on the user's machine was sometimes viewed as a security issue since it gave the user direct access to the runtime software, utilities included with it, and the application(s).

To make end-user machine administration easier, some organizations elected to not install the software locally and instead took advantage of remote installations and remote access across the network. Although the concept of remote access worked in many cases, it too came with issues, not the least of which was that Oracle did not consider this a supported configuration.

FSAL takes the best of both concepts; client/server and web. Using FSAL can provide the appearance and power of a native application, very much like older client/server Forms. The application runs in its own window, unlike when running a form in a browser. As a result, there is no risk of accidentally navigating away from the running form by pressing the Back or Forward button, bookmark, etc. With FSAL, the application will be hosted on a centralized application server (e.g. WebLogic Server) just as in earlier web deployed Forms versions. This means the Forms application modules are securely stored on this remote middle tier server. Although the user will not have direct access to the Forms modules (e.g. FMX, MMX, PLX, etc) which are on the remote server, they will be able to run these applications using a typical URL previously used when running a form in the browser.

The use of FSAL supports all the same functionality found when running a form in the browser, except event driven single sign-off. Single sign-on support was added in 12.2.1.4.0. JavaScript integration support for FSAL does not exist by default. However, JS integration can be enabled with a provided add-on (WJSI) in 12.2.1.3.0+ and a third party library (Eclipse/Jetty).

Requirements

The Forms Standalone Launcher

As mentioned, the FSAL feature is only available in Forms 12.2.1.0 and newer. The configuration requires that a small JAR file (frmsal.jar) be stored on the end-user machine. The file can be transferred to the end-user machine using any desirable method (e.g. web download, email, ftp, etc). The file can be stored anywhere on the user's machine as long the user has access to that directory and file. Storing it in the user's home directory is recommended, but not required.

The frmsal.jar file is the Forms Standalone Launcher. It is version specific; therefore it is not supported to use frmsal.jar from one installation with another. A checksum is also used to help ensure that the file is properly matched to the server and its Forms' version. This checksum is also helpful in ensuring that the launcher (frmsal.jar) has not been accidentally or maliciously replaced. As a result, you cannot use frmsal.jar that was downloaded from a Windows server and use it to run an application against a UNIX server or visa-versa. The launcher is not client platform specific. It can be used on Windows, Unix/Linux, or Apple Mac. Essentially, any platform that supports running Oracle Java.

All installations come with a usage page that illustrates how to use the launcher, as well as providing a download link for the frmsal.jar file. The page can be accessed using a URL like this:

```
http://example.com/forms/html/fsal.htm
```

Java

FSAL is a Java application and therefore Oracle Java is required on the user's machine. However, unlike when running a Forms application with the browser where a specific Java type (e.g. JRE – Plug-in) is required, FSAL supports using any Oracle Java distribution that supports running a Java application and was certified with your version of Forms. Depending on the user's platform, there are several possible distribution types that can be used. Currently, the following distributions are available for most common end-user platforms and can be used to run FSAL: JRE, JDK, and Server JRE. Java 11 on offers JDK at this time. Depending on the application's needs, which distribution you choose may vary. It is recommended that the contents of the Oracle Java installation you choose be carefully reviewed and/or your application be thoroughly tested before moving to production. Here is an overview of each Java distribution.

- JRE (applies to Java 8 only)

  The JRE installs most components needed for typical end-users. It includes everything needed to run a local java application, as well as the Java deployment components (i.e. Java Web Start and Java Plug-in). Java deployment components may not be available in releases newer than Java 8.

- JDK

  The JDK is generally for Java developers and includes far more than a typical user will need. The JDK includes a complete JRE plus tools for developing, debugging, and monitoring Java applications.

- Server JRE (applies to Java 8 only)

  The Server JRE is primarily for deploying Java applications on servers. It includes tools for JVM monitoring and tools commonly required for server applications, but does not include browser integration (the Java Plug-in), Java Web Start, auto-update, nor an installer. This distribution is delivered as a zip file that must be manually extracted. Because this distribution does not perform a software installation and is much more lightweight than the JRE and JDK distributions, this option is ideal for the FSAL when used along with a customized startup script or similar.

Be sure to check the Fusion Middleware Certification Guide to ensure that the version used is one that has been certified for use with the Forms version you are using.

Running an Application with FSAL

The Basics

FSAL, like a browser, needs to know the location of the application you want to run. In a browser, hyperlinks are sometimes used to launch other web pages or even Forms applications. Because FSAL is fully removed from a browser, you cannot easily use a web page to launch an application hosted by FSAL. So, knowing the complete Forms application URL is necessary. Currently, the use of URL redirects or rewrites is not supported, but may be technically possible depending on the server configuration. FSAL expects to receive a fully qualified URL that points to the Forms environment. A desktop shortcut or script/batch file can be used in place of a hyperlink in order to make starting the application simple and less error prone.

Running an application with FSAL is easy once the above requirements have been met. To start an application with FSAL, do the following:

1. Open a shell (e.g. DOS on Microsoft Windows) and verify the desired Java version is found.

   ```
   java –version
   ```

   The result should indicate the desired Java version. If not, the system's PATH may not be properly set.

2. Change directories to the user's home directory (assuming frmsal.jar was stored in the user's home directory) and execute the following command. Substitute your server in the example below. If the server is not running SSL enabled and instead is using HTTP, you can omit including "http://" from the URL entry, as it will be assumed.

```
java -jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"
```

   The application associated with the configuration section titled **`[standaloneapp]`** will run. You can use any configuration section as long as it contains the entries found in the [standaloneapp] example provided.

3. Output typically seen in the Java Console will appear in the shell used to start the application. If the "**`javaw`**" command is used rather than "**`java`**", a console will not be shown. If using the "**`java`**" command, closing this shell will terminate the application and therefore it should remain open through the life of the session. This behavior can be altered to accommodate your needs using various shell commands and associated switches. Refer to the operating system documentation for information on using the command shell on your platform.

These basic details can be used to create a desktop shortcut, batch script file, or custom executable that can be used to make launching the application easier. Here are the FSAL specific command line arguments:

-url *URL (required)*
URL is the fully qualified web address to the Forms environment, to include the configuration name. If config is not included, the default configuration will attempt to load. The URL should be quoted.

-t *time (optional - default value 60000ms)*
The time is the amount of time the launcher should wait for the server to provide its initial response before timing out. The value should be entered in milliseconds and be whole numbers only.

-showConfig *boolean (optional - default value FALSE)*
Display the config parameters received from the server in the console.

File Caching

Similar to the Java Plugin or even a browser, FSAL attempts to cache (store locally) files that may be reused the next time an application is launched. When starting FSAL from a Unix shell or DOS, the directory where files will be stored is displayed in the shell output during the loading process. Also displayed is whether the files are being downloaded from the server or reused from the existing cache. In the case where the files are downloaded from the server, text similar to the following will be shown:

Inspecting archive files in cache directory C:\Users\jdoe\AppData\Local\Temp\frmsal\example.com\12.2.1.4
Downloading archive file frmall.jar to cache subdirectory 8ymuqdqvdfe13a0d5vvema0dh

In the case where the files previously cached will be used, text similar to the following will be shown:

Inspecting archive files in cache directory C:\Users\jdoe\AppData\Local\Temp\frmsal\example.com\12.2.1.4
Using cached archive file frmall.jar from cache subdirectory 8ymuqdqvdfe13a0d5vvema0dh

To control the location of the cache files, change the location of tmpdir when starting the application. Consider the following example for Microsoft Windows users. Note that it uses the value of the Windows system variable USERNAME as the first level directory. This approach allows each user to have their own cache directory, assuming each user logs on to the machine as a different user.

```
java -Djava.io.tmpdir=D:/%USERNAME% -jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"
```

When the user is on a shared Unix/Linux platform this may be helpful because the downloaded cached files will be owned by the first user who ran the application. As a result, subsequent users may not have sufficient permissions to overwrite the old cached files with new ones. By creating unique locations for each user this problem can be avoided.

```
java -Djava.io.tmpdir=/scratch/$user/tmp -jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"
```

For some applications, it may not be desirable to use cached files, but instead always download from the server. This may also be true when troubleshooting technical issues. To disable file caching, use Fusion Middleware Control (FMC) to edit the FSAL template file basesaa.txt or webutilsaa.txt, based on your application. In the configuration, add the following line:

ignoreSaaCache=%ignoreSaaCache%

In the Forms Web Configuration (formsweb.cfg), add a new parameter *ignoreSaaCache* to the "standaloneapp" configuration or whichever configuration you are using to run the application. Set its value to **TRUE**. Cached files will now be ignored and downloading files from the server will occur each time the application is started.

Running Audio Enabled Applications with Java 11+

Beginning with Forms 12.2.1.4, Java 11+ (LST) can be used to run FSAL. However, Java 11 and newer no longer include JavaFX. JavaFX is necessary for the audio feature in Forms to function. If the applications do not use the Forms audio feature, these steps are not necessary. To enable JavaFX the following steps would be necessary on the user's machine.

1. On the user's machine, download and extract the JavaFX SDK from Gluon (not affiliated with Oracle) into a directory that the user has runtime (read, execute) permissions. Download the version that most closely matches the Java version used to run FSAL. JavaFX is now an Open Source project. However, be sure to review the Terms of Use and support options before downloading.

   https://gluonhq.com/products/javafx

2. Alter the typical FSAL startup command to include JFX.

   ```
   java --module-path C:/javafx-sdk-11.0.2/lib --add-modules=javafx.media,javafx.swing
   -jar frmsal.jar -url "https://example.com/forms/frmservlet?config=standaloneapp"
   ```

Running Behind a Proxy Server

In many cases, users will access a Forms application while within a corporate network. In some cases, this means that the user's machine requires the appropriate proxy configuration in order to access both internal and external content. In the case of using FSAL, the browser and system level settings may not be visible to the shell that launches the application. Therefore, it may be necessary to include such settings at the time FSAL is run. Because FSAL is a Java application, it is Java (e.g. java.exe) that must be aware of the needed proxy settings. There are several ways in which to inform Java of these settings.

If you want to use your system settings, do something like the following:

```
java -Djava.net.useSystemProxies=true –jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"
```

The use of this Java option (**-Djava.net.useSystemProxies=true**) will cause Java to attempt the call using the setting provided at the system level. It should be noted that this method is may not be supported when the system is configured to use Automatic Configuration Scripts (e.g wpad.dat).

Alternatively, proxy settings can be specifically included. Here are two examples:

```
java -Dhttps.proxyHost=example.com -Dhttps.proxyPort=80 -
Dhttps.nonProxyHosts="localhost|example.com" –jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"

java -Dhttp.proxyHost=example.com -Dhttp.proxyPort=80 -
Dhttp.nonProxyHosts="localhost|example.com" –jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"
```

Security Tips

The security of an application, the data exchanged within it, and the network on which it is hosted should be considered one of the most important aspects of any application deployment. A weakened security layer can be responsible for sensitive data breaches and malicious system attacks. As an application developer, DBA, System Administrator, or any other role in IT, it is your responsibility to ensure that proper security efforts are being used to protect the applications, the data, and their hosting systems.

This section will offer several tips that can be used to help improve the security related to using the FSAL. It is important that you use these tips only as examples. It is your responsibility to understand the concepts and research any that you do not fully understand. Improperly implementing any security configuration may put your system at risk. So, carefully review and test your changes before assuming they are correct. The tips provided here may not be unique to FSAL or even Forms. They are standard suggestions that relate to the technologies in use. Therefore, the availability of additional information is widespread. Do not use this paper as the only source of information for securing your application, its data, and its environment.

Be sure to also review the Additional Reading section included at the end of this document.

Secure Socket Layer

Secure Socket Layer or SSL (TLS – Transport Layer Security) is a cryptographic protocol used to provide encrypted communication between a source and destination of network traffic. This protocol works by creating a trusted connection, which is most often established by a public and private key exchange. The details of how SSL/TLS works is beyond the scope of this document.

Using SSL/TLS to run any application communicating on the network is very important with regard to securing data. The use of SSL/TLS for running any and all applications should be considered a requirement and not optional. Running an application through FSAL requires that the SSL/TLS certificate public key be included in the Java keystore. This may require the certificate(s) be manually imported if it is not already included in the keystore or provided by a known Certificate Authority (e.g. Symantec, Entrust, Comodo, etc). Specifically, this key must be imported into the Java keystore that is being used to run the form. In some cases, more than one certificate may

need to be imported, as a chain of certificates may exist. If you attempt to run an application using SSL and the needed certificate is not found or was improperly imported, a Java error similar to one of the following likely will be presented:

java.security.cert.CertificateException: No subject alternative names present …

OR

java.security.cert.CertificateException: No name matching example.com found …

OR

javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: …

Following one of the above errors will be an FSAL error.

FRM-92490: Unable to fetch applet parameters from server.

Steps for Configuring SSL/TLS with FSAL

In order to use SSL/TLS with FSAL, follow these steps prior to attempting the use of FSAL.

1. Obtain an SSL/TLS certificate from a known Certificate Authority and configure Oracle HTTP Server (or WebLogic Server) according to the instructions provided by the documentation for those components.
2. The public key portion of the certificate will be needed on the user's machine. If you do not have the public key as provided by your Certificate Authority, there are several ways to obtain it. Obtaining the key chain can be done from any machine that has access to the server. Here is an example of how to obtain the key(s) using the **openssl** command. This is pre-installed on most Unix/Linux platforms, but can also be obtained for Microsoft Windows.

   ```
   openssl s_client -showcerts -connect example.com:4443 > output.txt
   ```

   In the above example, replace **example.com:4443** with your server and SSL/TLS port number. The result of running this command will go into a file named "output.txt". Within the file you will see one or more certificates listed. The certificate is the contents between and including the BEGIN and END header/footer. Copy each certificate to its own file. Be sure to only include the BEGIN and END text exactly as shown below and the contents between them. Do not include any extra lines above or below the **-----BEGIN CERTIFICATE-----** or **-----END CERTIFICATE-----** when saving the file. Here is an example:

   ```
   -----BEGIN CERTIFICATE-----
   MIIFKDCCBBCgAwIBAgIBPTANBgkqhkiG9w0BAQsFADCBrjEpMCcGA1UEAxMgTmlj
   Y2subWFuc0BvcmFjbGUuY29tMRAwDgYDVQQLEwdTdXBwb3J0MQ8wDQYDVQQKEwZP
   cmFjbGUxGTAXBgNVBAcTEENvbG9yYWRvIFNwcmluZ3MxETAPBgNVBAgTCENvbG9y
   . . .
   -----END CERTIFICATE-----
   ```

   Alternatively, run the following to directly generate the needed certificate file.

   ```
   openssl s_client -showcerts -servername example.com -connect
   exaample.com:4443 | openssl x509 -outform PEM > cert.cer
   ```

3. Import the public key(s) into the Java keystore on the user's machine. If it is a certificate chain, be sure to import all keys in the chain (etc. signer, intermediate, root, etc). This means it may be necessary to run the appropriate utility more than one time. To import the certificate public key(s) you can use the Java **keytool** utility which is included in all Oracle Java distributions. Alternatively, you can use one of the many free utilities available on the Internet.

   Here is an example of how each certificate could be inserted manually. Refer to the Java documentation for details on how to use the **keytool** utility.

   ```
   keytool -import -alias <server_name> -keystore
   <JAVA_HOME>/jre/lib/security/cacerts -file cert.cer
   ```

4. Run FSAL with SSL, as shown in the example provided in the previous section of this paper, "*Running an Application with FSAL*".

Troubleshooting SSL/TLS

TIP 1:
Verify the correct keystore was updated. The trusted keystore is found in JRE_HOME/lib/security or in the user's home directory. The default file name is **cacerts**. The keystore name will differ if you created your own. If you created your own, it would have been put in the location you provided at that time. If you did not provide a location it will either be in the default location previously mentioned or in the user home. Verify that the file modified date/time is the same as when you executed the import command.

TIP 2:
Verify that you have imported all the certificates in the chain. If you know there are three, list all the certificates and verify that all three are shown. Here is an example:

```
keytool -list -keystore "C:\java\jdk1.8.0_221\jre\lib\security\cacerts"
```

TIP 3:
If the keystore you updated is not in the JRE_HOME or user home directory, copy it to the JRE_HOME. Be sure to create backup files of any files you may be overwriting.

TIP 4:
Enable Java SSL/TLS debugging. To enable debugging mode, run FSAL and the desired form as follows:

```
java -Djavax.net.debug=all -jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"
```

The output can also be redirected to a text file as follows (assumes Microsoft Windows):

```
java -Djavax.net.debug=all -jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp" >
C:\existing_directory\output.txt
```

Signed Code

If you electronically send someone an applet or application to run, the recipient needs a way to verify that the code came from you and was not modified in transit (for example, by a malicious user intercepting it). Signing code with a digital signature helps to ensure such security of the files sent.

Forms, Java Plug-in, and Java Web Start

To run a Forms application from a browser with the Java Plug-in or Java Web Start, the user likely clicks on a hyperlink or bookmark or manually enters a URL into a browser. Unrelated to the use of Oracle Forms, the simple fact that the user is using a browser puts their system at potential risk. With so many malicious web sites accessible on the Internet, stumbling upon a malicious site, intentionally or unintentionally is almost inevitable. Fortunately, the Java Plug-in and Java Web Start have special security built-in to help mitigate the chances of malicious Java applications being run on the user's machine. One of these key security features is a requirement to only permit signed (and trusted) applications to be run. Unsigned applications are no longer permitted and are blocked. Since there is cost and organizational verification associated with obtaining a trusted code signing certificate (digital signature), attempts to deliver malicious content in this manner are fairly rare.

All Oracle Forms provided Java JAR files that run on the user's machine are signed in order to comply with Java's security requirements mentioned above. This allows Forms applications to safely run in a browser. It is further required that custom JAR files (those not provided by Oracle) also be properly signed with a trusted certificate. Using self-generated certificates is no longer supported and should be avoided. Although using self-generated certificates may seem to function, support for this functionality may be removed from Java in the future.

FSAL and Signed JARs

Proper code/JAR signing is strongly recommended even when using FSAL. However, because applications launched using FSAL do not launch from a browser, the risk of running un-trusted Java code is greatly reduced. Although running native Java applications do not have the same level of built-in security found in the Plug-in and Web Start, some signed code verification can and should be enabled.

To enable signed code awareness for FSAL and specifically your application, following these steps:

1. Obtain the public keys associated with all JAR files used in your application. Remember that Oracle provided JARs are signed when delivered and all have the same certificate. Therefore, only one public key will be needed for Oracle's files unless you have resigned them or obtained patches from Oracle Support that may have replaced these files from their originals. If you don't have the signers public key for the JAR files used by your application, you can use the following Java command for each of the JARs to collect the needed information. The first certificate returned is the signer certificate and the one you will need for the steps that follow.

   ```
   keytool -printcert -rfc -jarfile yourJarFile.jar
   ```

2. On the user's machine, import each public key (from JARs used in the application) into the Java keystore that will be used at runtime.

   ```
   keytool -importcert -alias foo -file C:\foo.pem
   ```

   The above alias can be any alphanumeric string, however be sure to remember it as it will be needed in a coming step. Also, because the –keystore switch was not included, a keystore in the user's home directory will be used (created if it doesn't exist). The file will be named ".keystore". To use a different keystore and/or keystore name, add the **`-keystore`** switch as desired. Repeat the above step for each certificate that you have to import.

   Refer to the Additional Reading section of this document for helpful links to using the "keystore" utility.

3. If one does not exist or if you want a fully customized file, create a **.java.policy** file in the user's home directory. Otherwise, use the default file. <JAVA_HOME>\jre\lib\security\java.policy Add the following to it:

```
keystore ".keystore";
grant signedBy "foo" {
   permission java.security.AllPermission;
};
```

The "foo" reference is a pointer to the alias used in Step 2. Add additional "grant signedBy" sections for each key that was imported in Step 2. The value of keystore is a pointer to the keystore file you expect to use.

4. Run your Forms application using FSAL with this slightly modified command entry:

```
java -Djava.security.manager -jar frmsal.jar -url
"https://example.com/forms/frmservlet?config=standaloneapp"
```

When running the application, JAR certificates will be matched with those in the keystore. If they do not match, it will not run. In the above example, it is assumed that the default **java.policy** is used or a **.java.policy** was created in the user's home directory. You can alternatively access a custom policy file from an alternative directory on the user's machine or a remote location (URL).

```
java -Djava.security.manager -Djava.security.policy=${user.home}/foo.policy
-jar frmsal.jar -url "https://example.com/forms/frmservlet?config=standaloneapp"
```

```
java -Djava.security.manager -Djava.security.policy=https://example.com/foo.policy
-jar frmsal.jar -url "https://example.com/forms/frmservlet?config=standaloneapp"
```

To ensure users are taking advantage of this validation, a startup script or similar should be used. Using a script will help prevent the possibility of typographical errors or the possibility of not using the security manager.

Single Sign-on

Single Sign-on is an authentication concept that offers many valuable advantages to web applications. The idea of integrating Forms with Single Sign-on (SSO) is often seen as having at least two significant benefits over using the Forms database login functionality. The first is that the user only has to be authenticated to an application one time while their browser remains open. This authentication can be shared with other browser applications or other Forms applications. This means the user would not be required to login to each application, but rather only the first. This applies to all applications protected by the same SSO server and applications running or launched from the same browser session. The second advantage is that the database credentials for running Forms applications would be hidden from the user.

To use SSO with FSAL you must perform the same configuration steps needed for using SSO with Forms in any other case. For information about how SSO is configured with Forms and how it works, please refer to the "Working with Oracle Forms Guide". Once this common configuration has been completed, add **ssoMode=%ssoMode%** to the Forms template files basesaa.txt and webutilsaa.txt, using Fusion Middleware Control. To enable SSO for a specific application, set **ssoMode=true** in the desired application configuration of the Forms Web Configuration settings associated with the application to be protected.

FSAL SSO Servlet Parameters

The following Forms Web Configuration (aka formsweb.cfg) Servlet parameters are used with FSAL only. By default, the Forms Web Configuration does not include any references to these entries. Add them manually as needed. Be sure to make all configuration changes in Fusion Middleware Control. More information about applet (and Servlet) parameters and how to use them can be found in the "Working with Oracle Forms Guide".

Applet Parameter	Description	Applies To
ssoSaaBrowserLaunchTimeout	Specifies, in seconds, how long the Forms servlet will wait for the initial request from the browser that was launched by FSAL for SSO authentication. If the interval expires, the fatal error FRM-93249 will be reported. Valid values: Integers in the range 1-300. Default: 15	Servlet
ssoSaaBrowserPageTimeout	Specifies, in seconds, how long the Forms servlet will wait for the user to enter data into a browser page during SSO authentication for an FSAL application. If the interval expires, the fatal error FRM-93382 or FRM-93383 will be reported. Valid values: Integer >= 15, or 0 (wait indefinitely). Default: 0	Servlet
ssoSaaWaitInterval	Specified the interval, in seconds, at which FSAL reissues requests to the Forms servlet while SSO authentication is proceeding in the launched browser window. Larger values reduce network traffic but increase the chances of an intermediate agent timing out (thereby producing the fatal error FRM-93248). Valid values: Integer >= 5, or 0 (do not reissue requests). Default: 25	Servlet
ssoSuccessLogonURL	The URL to redirect to if SSO authentication completes successfully for an FSAL application.	Servlet

Resources

http://www.oracle.com/technetwork/developer-tools/forms/documentation

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

https://docs.oracle.com/javase/tutorial/security/tour2/step2.html

https://docs.oracle.com/javase/tutorial/security/toolsign/rstep1.html

https://docs.oracle.com/javase/tutorial/security/toolsign/rstep2.html

https://docs.oracle.com/javase/tutorial/security/toolsign/rstep3.html

https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#Debug

https://docs.oracle.com/javase/8/docs/technotes/guides/security/PolicyFiles.html

https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html

https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/ReadDebug.html

https://docs.oracle.com/javase/8/docs/technotes/guides/net/proxies.html

https://blogs.oracle.com/jtc/installing-trusted-certificates-into-a-java-keystore

https://confluence.atlassian.com/kb/how-to-import-a-public-ssl-certificate-into-a-jvm-867025849.html

https://en.wikipedia.org/wiki/Transport_Layer_Security

https://www.digicert.com/ssl

https://en.wikipedia.org/wiki/Chain_of_trust

https://www.openssl.org

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Integrated Cloud Applications & Platform Services

USING AND SECURING APPLICATIONS RUNNING WITH FSAL
September 2019
Author: Oracle Product Management
Contributing Authors: Oracle Support Services

Oracle is committed to developing practices and products that help protect the environment